A black background with white text

Description automatically generated

**Projekt - Podstawy sieci neuronowych**

**Temat: Klasyfikacja binarna rezygnacji klientów za pomocą wielowarstwowej sieci neuronowej MLP**

Grupa numer: 3

Poniedziałek 17:05 – 18:45

Wykonał:

Michał Białek (264285)

Termin oddania projektu: 19.01.2025

# Wprowadzenie do projektu

Celem projektu jest zaprojektowanie, implementacja i wytrenowanie sztucznej sieci neuronowej do binarnej klasyfikacji rezygnacji klientów banku na podstawie dostępnego zestawu danych z platformy Kaggle.com. Projekt obejmuje pełny proces analizy danych, przygotowania modelu, dobór odpowiednich parametrów, i ich optymalizacja, oraz ewaluacji wyników badań

## Analiza problemu

Współczesne banki borykają się z problemem rezygnacji klientów z kont bankowych i jednoczesną rezygnacją z usług bankowych (eng. clients churn). Taka sytuacja jest niekorzystana dla banku, a nawet w skrajnym przypadku może skutkować upadłością. Rozwiązaniem tego problemu jest identyfikacja klientów, u których występuje wysokie prawdopodobieństwo rezygnacji z usług, oraz wdrożenie działań zapobiegawczych, takich jak spersonalizowane oferty, promocje, czy lepsza obsługa klienta.

## Cel projektu

Celem poniższego projektu jest zbudowanie na bazie dostępnych klienckich, modelu klasyfikacyjnego, który przewiduje rezygnację klienta z usług bankowych.

# Analiza danych wejściowych

Poniższy model bazuje na pliku Churn Modelling z strony Kagge.com – [link](https://www.kaggle.com/datasets/shrutimechlearn/churn-modelling)

Dane wejściowe są w formacie CSV, i zawierają 14 kolumn, które można skategoryzować na podstawie typu danych, oraz ich użyteczności w modelu:

## Klasyfikacja na podstawie przydatności danych

Z wszystkich 14 kolumn, jedynie 3 kolumny (RowNumber,CustomerId,Surname), nie przenoszą żadnych wartości potrzebnych do analizy odejść klientów, a pełnią jedynie rolę identyfikacyjną.

## Klasyfikacja na podstawie typu danych

Z podanych 11 kolumn można zaklasyfikować na podstawie typu przechowywanych danych:

6 kolumn o wartościach numerycznych: CreditScore (zdolność kredytowa) , Age (wiek posiadacza konta), Tenure (czas posiadania konta przez klienta), Balance (aktualne saldo na koncie bankowym) , NumberOfProducts (ilość produktów bankowych) , EstimatedSalary (Szacowana wartość wpłat).

2 kolumny o wartościach kategorycznych, które zostaną przekształcone na wartości binarne, lepsze do analizy, metodą One-Hot Encoding. Są to Geography (dane o kraju pochodzenia posiadacza konta), oraz Gender (płeć)

2 kolumny o wartości binarnej – HasCreditCard (posiadanie karty kredytowej), IsActiveMember (flaga oznajmiająca, czy podany użytkownik jest aktywny)

1 kolumna - zmienna docelowa – Exited (zawiera informację, czy klient opuścił bank)

# Analizy problemu i wyboru odpowiedniej architektury sieci neuronowej

Do problemu klasyfikacji binarnej, tego rodzaju najlepszym wyborem jest sieć wielowarstwowa (MLP - Multi-Layer Perceptron) z następujących powodów:

Sieć MLP obsługuje różnorodne dane wejściowe, od numerycznych, jak i również kategoryczne ( po przekształceniu przez One-Hot Encoding)

Zdolność do modelowania nieliniowych zależności miedzy cechami danych wejściowych modelu. Wiele cech, takich jak CreditScore, Balance, czy IsActiveMember, może mieć nieliniowy wpływ na odejście klienta. MLP dzięki swoim nieliniowym funkcjom aktywacji (np. ReLU, Sigmoid) jest w stanie uchwycić te zależności.

Jedną z kluczowych cech jest również możliwość dostosowywania ilości neuronów w poszczególnych warstwach, jak i również samą ilość poszczególnych warstw ukrytych.

## Opis struktury wielowarstwowej sieci MLP

Sieć MLP składa się z 3 warstw:

1. Warstwa wejściowa - przyjmuje dane wejściowe w postaci wektora cech. Liczba neuronów w tej warstwie jest równa liczbie cech w danych wejściowych.
2. Warstwa ukryta - jest to jedna lub więcej warstw między warstwą wejściową a wyjściową. Każdy neuron w warstwie ukrytej oblicza liniową kombinację wag i przesunięć (biasów) na swoich wejściach, a wynik przepuszczany jest przez funkcję aktywacji (np. ReLU, sigmoid, tanh).

– są to wagi, które określają, jak ważne są poszczególne wejścia w procesie obliczania wyjścia. Wagi są dostosowywane podczas procesu uczenia się.

– reprezentują dane wejściowe, które mogą pochodzić z poprzedniej warstwy sieci lub bezpośrednio z danych treningowych

- to stała wartość dodawana do obliczeń, która pozwala neuronowi lepiej dopasowywać się do danych nawet wtedy, gdy wszystkie wejścia są równe 0. Bias umożliwia translację funkcji aktywacji.

- To wynik sumowania iloczynów wag i odpowiadających im wejść oraz biasu.

Oraz oznaczenia takie jak:

**Wyjście neuronu** () - To wynik działania funkcji aktywacji na sumę ważoną (z). Funkcja aktywacji wprowadza nieliniowość do modelu, co pozwala sieci neuronowej rozwiązywać złożone problemy.

**Funkcja aktywacji** - może to być np. funkcja sigmoidalna, ReLU (Rectified Linear Unit) lub inna funkcja nieliniowa, która przekształca wartość z w wartość wyjściową a.

1. **Warstwa wyjściowa - Generuje wyniki na podstawie obliczeń w warstwach ukrytych.**

Do warstwy wejściowej podawane są dane, i następuje proces uczenia:

* Sieć dostosowuje swoje wagi i biasy na podstawie błędu przewidywań w procesie propagacji wstecznej (backpropagation) oraz algorytmu optymalizacji (np. SGD, Adam).
* Celem jest minimalizacja funkcji kosztu cross-entropy dla klasyfikacji

# Rodzaj i dobór parametrów sieci, oraz ich wpływ na działanie sieci

Podczas tworzenia sieci neuronowych musimy dobrać odpowiednie parametry sieci, w celu uzyskania jak najlepszych wyników działania.

Do dyspozycji mamy następujące parametry podstawowe:

1. Liczba warstw ukrytych
2. Liczba neuronów w warstwach
3. Funkcja aktywacji

Oraz następujące parametry optymalizacyjne:

1. Szybkość uczenia ( Learning rate)- nieska wartość zapewnia stabilne, ale wolne uczenie. Z drugiej strony duża wartość może przyśpieszyć proces uczenia, ale zwięksa ryzyko braku zbieżności.
2. Batch size - Mały batch size prowadzi do szybszej aktualizacji wag, ale z większym szumem. Duży zapewnia bardziej stabilne uczenie, ale jest wolniejszy i wymaga więcej pamięci.
3. Liczba epok - Zbyt mała liczba epok prowadzi do niedouczenia, natomiast zbyt duża liczba epok może prowadzić do przetrenowania.

Ważnym aspektem jest również rekuraryzacja, czyli zapobieganie przeuczeniu. Do rozwiązania problemu przeuczenia, stosujemy techniki takie jak:

Dropout – czyli wyłączanie losowe 20-50% neuronów w warstwach ukrytych

Regularyzacja wag – czyli karanie modelu, za dawanie dużych wartości wag

Podczas trenowania modelu jest również możliwość monitorowania działania procesu uczenia, oraz bieżące reagowanie na zmiany parametrów modelu. Przykładem takiego mechanizmu może być early stopping, czyli mechanizm zatrzymujący trening, w przypadku wykrycia braku poprawy metryk

# Implementacja modelu sieci MLP, oraz obserwacyjny dobór optymalnych parametrów

Poniżej przedstawiam domyślną implementację sieci neuronowej wykonanej w języku Python, z wykorzystaniem następujący bibliotek:

1. Numpy - Biblioteka do obliczeń numerycznych. Wykorzystujemy funkcję seed, do ustawienia ziarna losowości.
2. Pandas – bibliteoka do analizy i manipulacji danymi. W naszym przypadku wykorzystujemy do odczytywania danych z pliku csv, manipulowanie danymi, oraz do kodowania zmiennych kategorycznych metodą one-hot encoding
3. Seaborn – bibliteka do wizualizacji danych (Wizualizacja macierzy konfuzji w formie mapy cieplnej)
4. Matplotlib – wizualizacja danych
5. Scikit-learn - Biblioteka do uczenia maszynowego, zawiera narzędzia do preprocessingu danych, podziału zbiorów danych, trenowania modeli i ewaluacji. Dzięki tej bibliotece dzielimy dane na zbiory treningowe, walidacyjne i testowe.
6. TensorFlow – tworzenie i trenowanie sieci neuronowych. Używamy w niej funkcji do tworzenia i definowania parametrów sieci neuronowej.

Pierwotna implementacja posiada następujące parametry:

* 3 warstwy ukryte, o ilości neuronów odpowiednio 128,64,32 neurony, wraz z funkcją aktywacji ReLU.
* Wartość Drouput ustawioną na 0,5 oraz Regularyzacji Lambda = 0.001
* Wartość wyjściowa to neuron z aktywacją sigmoid
* Dodatkowo zastosowano mechanizm, binary\_crossentropy, który nagradza i karze za dobre/złe predykcje

Podstawowa, jak i również różne implementacje podanej sieci neuronowe jest dostępna na repozytorium github:

<https://github.com/MichalBialek01/neural-networks-churn-detection>

Kroki opisujące implementację bazową:

1. Wczytanie pliku „Churn\_Modelling.csv” do obiektu typu DataFrame
2. Wykonujemy podstawową walidację danych, czyli wyświetlamy informację na temat sieci neuronowej, oraz sprawdzamy, czy pobrany dataset posiada wartości puste. W przypadku, gdyby wystąpiły, potrzebna byłaby walidacja danych i odrzucanie przypadków, gdzie dane byłby niepoprawne. W tym kroku wykonujemy również obserwacje zbilansowania wartości Exited poprez sns.countplot. W przypadku, gdy wartości słupków są na podobnym poziomie, oznacza to, że dane są zbilansowane. W przypadku danych nie zbilansowanych, należałoby dane poddać preprocesowaniu

W naszym przypadku mamy niezbilansowany rozkład:  
A graph with a bar

Description automatically generated with medium confidence

W tym przypadku, z tego powodu idzie zauważyć dysproporcje. Z tego powodu należy dodać technikę nadpróbkowania (oversampling) dla klasy mniejszościowej (dla nas 1) na zbiorze treningowym. Pozwala to zwiększyć sztucznie ilość próbek. Do tego można zastosować RandomOverSampler z imblearn.over\_sampling

1. Usunięcie niepotrzebnych kolumn ('RowNumber', 'CustomerId', 'Surname')
2. Kodowanie zmiennych kategorycznych

Kolumnę Gender mapujemy, na wartości binarne, żeby w przypadku wartości Male, wynikiem była wartość 1, oraz gdy wartość jest równa Female wartość wynosiła 0.

Wykonujemy również One-Hot Encoding kolumny Geography, przy pomocy biblioteki pandas

1. Dokonujemy podziału na cechy X (dane wejściowe) , oraz etykiety Y (zmienna docelowa)
2. Dzielimy zbiory na zbiór: treningowy, walidacyjny i testowy w proporcjach:  
   70% dane treningowe, 15% dane walidacyjne oraz 15% dane testowe
3. Dokonujemy standaryzacji danych.

Standaryzacja danych jest stosowana, aby przyspieszyć proces uczenia i poprawić jego stabilność. Ujednolica ona skalę cech (średnia 0, odchylenie standardowe 1), dzięki czemu żadna z nich nie dominuje nad innymi. Dodatkowo zapobiega problemom z gradientami (eksplozją lub zanikiem) oraz poprawia działanie funkcji aktywacji, takich jak sigmoid czy ReLU. Dzięki standaryzacji algorytm uczy się szybciej i dokładniej.

W kodzie wykorzystujemy StandardScaler do standaryzacji.

1. Tworzymy uproszczony model sieci neuronowej, o określonych parametrach:  
   W podstawowym przypadku 3 warstwy o liczbie neuronów 128,64,32. Podajemy liczbę cech wejściowych, wartość regularayzacji L2, rodzaj funkcji aktywacji, oraz dopout – wszystkie te parametry, dla każdej warstwy. Ostatnim krokiem jest dodanie warstwy ostatniej z jednym neuronem z aktywacją signoid.
2. Kompilujemy model, podając optymalizator Adam, z ustalonym learing\_rate na wartość 0.001, oraz binary\_crossentropy.
3. Trenowanie modelu – w naszym przypadku stosujemy mechanizm nadzorowania EarlyStopping, który śledzi wynik błędu na zbiorze walidacyjnym, z ustawioną wartością patience=10, co oznacza, że jeżeli wynik walidacyjny nie ulegnie poprawie, trenowanie zatrzyma się. Trenowanie odbywa się poprzez model.fix, w której podajemy parametry takie jak: liczba epok=50, rozmiar paczki danych na jedną iterację -32
4. Następuje trenowanie modelu

## Ewaluacja modelu

Po procesie trenowania modelu liczymy metryki takie jak:

1. **test\_loss** – finalna wartość funkcji kosztu na zbiorze testowym.
2. **test\_accuracy** – końcowa dokładność na zbiorze testowym.
3. **roc\_auc\_score** – miara AUC-ROC (Area Under Curve – Receiver Operating Characteristic). Im bliżej 1, tym lepszy model.
4. **f1\_score** – łączy precyzję i czułość w jedną liczbę. Szczególnie użyteczne w przypadku niezbalansowanych danych.

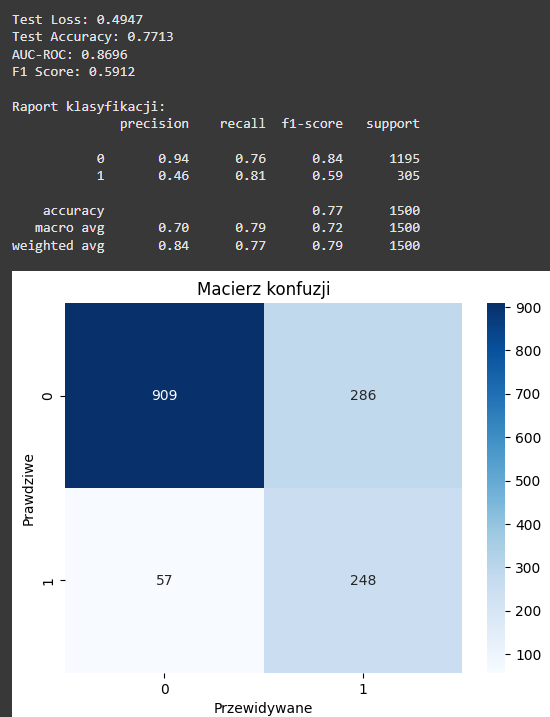
Ostatnim krokiem, który pozwoli nam zwizualizować działanie sieci jest macierz konfuzji. Jest to macierz 2x2, która służy do oceny wyników klasyfikacji. Każdy element macierzy ma odpowiednie znaczenie:

True Positives (TP): Liczba przypadków, w których model poprawnie przewidział klasę pozytywną (np. przewidziano 1, a rzeczywista etykieta to 1)

True Negatives (TN): Liczba przypadków, w których model poprawnie przewidział klasę negatywną (np. przewidziano 0, a rzeczywista etykieta to 0).

False Positives (FP): Liczba przypadków, w których model błędnie przewidział klasę pozytywną (np. przewidziano 1, ale rzeczywista etykieta to 0).

False Negatives (FN): Liczba przypadków, w których model błędnie przewidział klasę negatywną (np. przewidziano 0, ale rzeczywista etykieta to 1).

W naszym przypadku bazowym, wyniki wychodzą następujące:  


# Testy w zależności od parametrów:

Parametry, które warto zmienić, aby zbadać zachowanie modelu, w zależności od parametru, włączenia/wyłączenia optymalizatora/nadzorowania.

* **Liczba warstw**: od 1 do 3 (lub nawet 4).
* **Liczba neuronów**: 16, 32, 64, 128 (ewentualnie 256 dla dużych sieci).
* **Funkcje aktywacji**: ReLU, tanh, softplus.
* **Optymalizator**: Adam (z domyślnymi parametrami) oraz np. SGD (z momentum).
* **Liczba epok**: wybrane poziomy (np. 20, 50, 100).
* **Batch\_size**: 32, 64 (opcjonalnie 16, 128).
* **EarlyStopping**: włączone / wyłączone.

**Test1:**

**- 1 warstwa ukryta (32 neurony)**

**- Aktywacja: ReLU**

**- Optymalizator: Adam**

**- Liczba epok: 20**

**- batch\_size: 32**

**- EarlyStopping: wył**

**A screenshot of a computer

Description automatically generated**

**Przypadek 2:**

**- 1 warstwa ukryta (32 neurony)**

**- Aktywacja: ReLU**

**- Optymalizator: Adam**

**- Liczba epok: 50**

**- batch\_size: 32**

**- EarlyStopping: włączone**

**A screenshot of a computer

Description automatically generated**

**"""**

**Przypadek 3:**

**- 2 warstwy ukryte: (64, 32)**

**- Aktywacja: ReLU**

**- Optymalizator: Adam**

**- Liczba epok: 50**

**- batch\_size: 32**

**- EarlyStopping: włączone**

**"""**

**A screenshot of a computer

Description automatically generated**

Przypadek 4:

"""

Przypadek 4:

- 2 warstwy (64, 32)

- Aktywacja: ReLU

- Optymalizator: SGD (momentum=0.9)

- Liczba epok: 50

- batch\_size: 32

- EarlyStopping: włączone

"""

A screenshot of a graph

Description automatically generated

"""

Przypadek 5:

- 2 warstwy (64, 32)

- Aktywacja: tanh

- Optymalizator: Adam

- Liczba epok: 50

- batch\_size: 32

- EarlyStopping: włączone

""" A screenshot of a computer

Description automatically generated

Przypadek 6:

- 3 warstwy (128, 64, 32)

- Aktywacja: ReLU

- Optymalizator: Adam

- Liczba epok: 50

- batch\_size: 32

- EarlyStopping: włączone

"""

A screenshot of a computer

Description automatically generated

"""

Przypadek 7:

- 3 warstwy (128, 64, 32)

- Aktywacja: tanh

- Optymalizator: Adam

- Liczba epok: 100

- batch\_size: 32

- EarlyStopping: włączone

"""

A screenshot of a computer

Description automatically generated

"""

Przypadek 8:

- 2 warstwy (64, 32)

- Aktywacja: softplus

- Optymalizator: Adam

- Liczba epok: 50

- batch\_size: 64

- EarlyStopping: włączone

"""

A screenshot of a computer

Description automatically generated

"""

Przypadek 9:

- 3 warstwy (64, 32, 16)

- Aktywacja: ReLU

- Optymalizator: SGD (momentum=0.9)

- Liczba epok: 50

- batch\_size: 32

- EarlyStopping: wyłączone

"""

A screenshot of a computer

Description automatically generated

Przypadek 10:

- 3 warstwy (128, 64, 32)

- Aktywacja: softplus

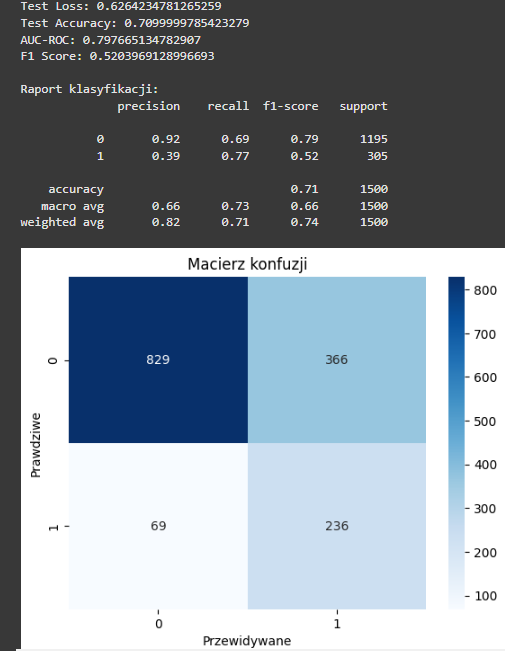
- Optymalizator: Adam

- Liczba epok: 50

- batch\_size: 32

- EarlyStopping: włączone

"""



Przypadek 11

- 3 warstwy (128, 64, 32)

- Aktywacja: softplus

- Optymalizator: SGD (z momentum=0.9)

- Liczba epok: 50

- batch\_size: 32

- EarlyStopping: wyłączone

"""

A screenshot of a computer

Description automatically generated

"""

Przykład 12:

- 3 warstwy (8, 4, 2) - bardzo małe warstwy

- Aktywacja: softplus

- Optymalizator: SGD (z momentum=0.9)

- Liczba epok: 50

- batch\_size: 32

- EarlyStopping: wyłączone

"""

Podsumowanie – wnioski i napotkane problemy

Sieci neuronowe umożliwiają rozwiązywanie skomplikowanych problemów z kategorii identyfikacji i predykcji.

Sieci możemy podzielić na wiele rodzajii, ze względu na charakter rozwiązywanego problemu (klasyfikacyjny, klasyfikacyjny-binarny, regresyjny), jak i również na architekturę.

Ważnymi aspektami pracy z sieciami neuronowymi jest odpowiednie rozpoznanie danych wejściowych, oraz zastosowanie pre-procesowania danych, która obejmuje usunięcie zbędnych wartości, weryfikację i ewentualne usunięcie wartości błędnych, kodowanie zmiennych kategorycznych, oraz standaryzację danych.

Jednym z ważniejszych kroków jest dobór odpowiedniej sieci neuronowej, do charakteru rozwiązywanego problemu, w celu jak najlepszego działania

Najważniejszym aspektem pracy z sieciami neuronowymi jest dobór odpowiednich parametrów sieci, w celu osiągnięcia możliwie jak najlepszych wyników. Oprócz doboru parametrów podstawowych, nowoczesne biblioteki umożliwiają optymalizację procesu uczenia, i doboru poprzez zastosowanie optymalizatora Adam, funkcję binary\_crossentropy, jak i również monitorowanie trenowanego modelu, i odpowiednie działania w przypadku braku poprawy metryk (np. straty walidacyjnej w nadzorowaniu EarlyStopping**)**

A screenshot of a graph

Description automatically generated